دانشکده مهندسی کامپیوتر

جزوه کارگاه

Blazor WebAssembly and Blazor Server

استاد درس: سیّد صالح اعتمادی

مدرس کارگاه: محمّد مهدی عبداللهپور

گردآورنده: روزبه غزوی

نیم‌سال دوم

سال تحصیلی ۹۸-۹۹

جلسه ۱

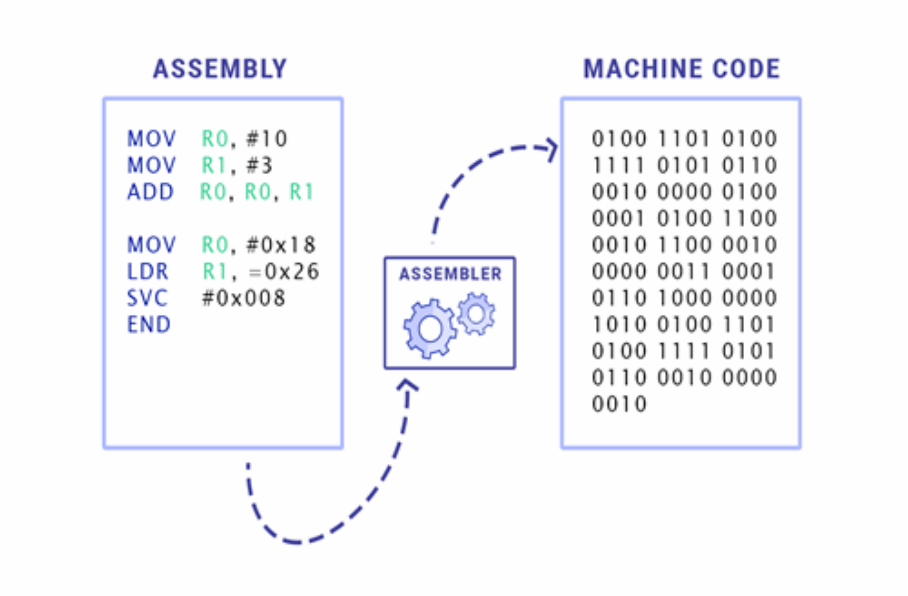
Introduction to WebAssembly

روزبه غزوی - ۱۳۹۸/۵/۲۱

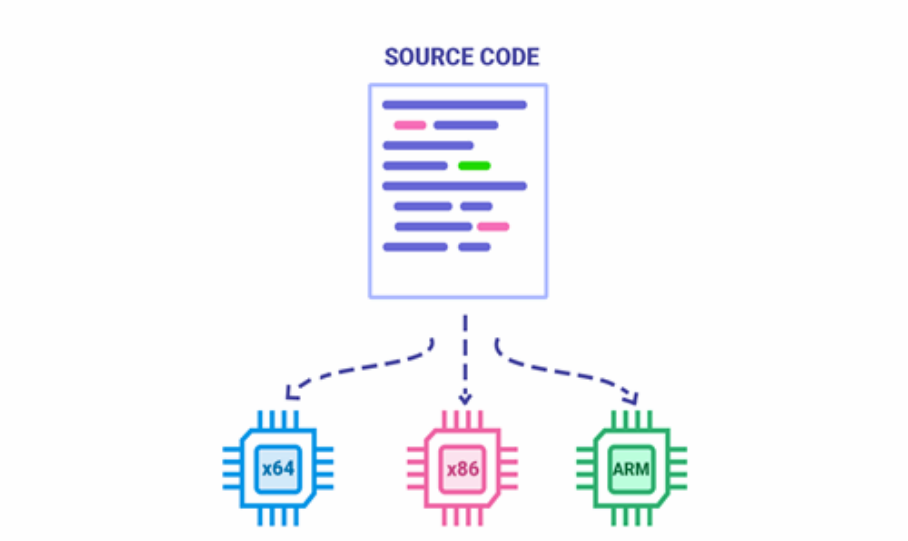
۱.۱ WebAssembly چیست؟

WebAssembly یک نوع جدید از کد است که در مرورگرهای جدید اجرا می شود و برای افزایش کارایی برنامه ها در وب ایجاد شده است. WebAssembly بصورت فرمت سطح پایین باینری می باشد و حجم کدهای آن کم است، بنابراین سریع بارگذاری و اجرا می شود. نیازی نیست که کد باینری بنویسیم بلکه بعد از نوشتن کد با زبان سطح بالا آن را به کد باینری کامپایل می کنیم.

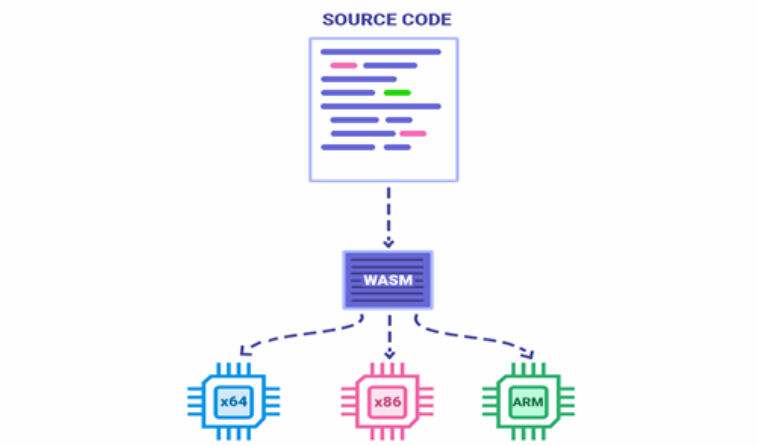
Assembly همان کدهای سطح پایین قابل خواندن توسط انسان است که خیلی به کد ماشین نزدیک است و کد ماشین همان اعداد دودویی است که پردازنده می فهمد. برای درک بهتر به تصویر صفحه بعد توجه کنید.



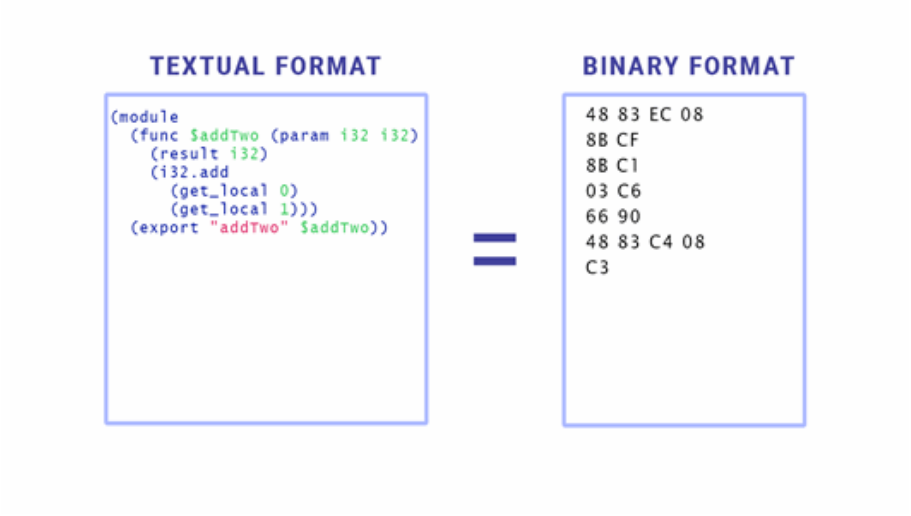
هر زبان سطح بالا برای اجرا در پردازنده، به زبان ماشین ترجمه می شود. پردازنده های دارای معماری های مختلف نیاز به ماشین کدهای خاص خود دارند.



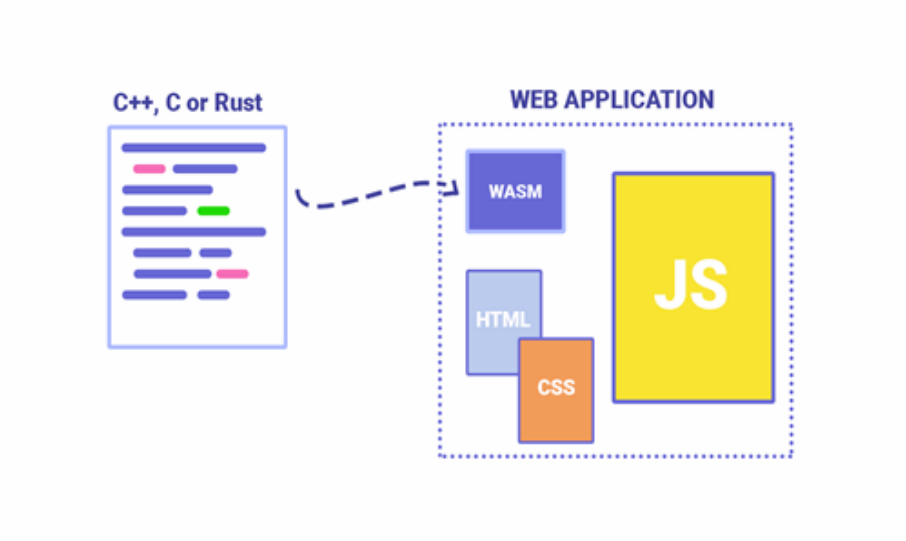
اما همانطور که از نام WebAssembly مشخص است، آن یک زبان Assembly نیست چون برای ماشین ها معنا ندارد و برای مروگرها است. هنگامی که کدهای WebAssembly را برای اجرا به مرورگر می دهیم دیگر نمی دانیم چه نوع ماشینی آن را اجرا خواهد کرد.



WebAssembly یک زبان برای ماشین های مفهومی است که مخرج مشترک سخت افزارهای رایج در دنیای واقعی می باشد و هنگامی که مرورگر کدهای WebAssembly را دانلود می کند، می تواند به سرعت آن را به زبان ماشین تبدیل کند. WebAssembly یک فرمت متنی دارد که نسبتا قابل خواندن است با پسوند wat و یک فرمت باینری دارد که به مرورگر تحویل داده می شود با پسوند .wasm



WebAssembly شما را قادر می سازد که توسط زبان های C C++، و Rust برنامه بنویسید و آن را به Module WebAssembly کامپایل نمایید. سپس آن را در مروگر بارگذاری کرده و توسط Javascript فراخوانی نمایید. بنابراین WebAssembly قرار نیست جایگزین Javascript شود بلکه قرار است با آن و در کنار آن کار کند.



۲.۱ چرا به WebAssembly نیاز داریم؟

نرم افزارهایی مثل بازی های رایانه ای، ویرایش ویدئو، رندر سه بعدی و یا تولید موسیقی را تصور کنید. این نرم افزارها محاسبات زیادی نیاز دارند و به درجه بالایی از کارایی احتیاج دارند، کسب این نوع از کارایی از Javascript سخت است.

Javascript به عنوان یک زبان اسکریپت نویسی ساده شروع به کار کرد که با اسناد HTML در تعامل بود. هدف اصلی طراحی آن سادگی یادگیری و سادگی نوشتن آن بوده و نه سرعت. در طی سال ها مرورگرها بهبودهایی در مفسر Javascript ایجاد کرده اند تا در کارایی آن تاثیرگذار باشد.

همزمان با بهبود کارایی Javascript، لیست چیزهایی که باید در مرورگر اجرا شود گسترش یافته است. هایAPI جدید نیازمندی های جدیدی را به همراه داشته اند از جمله تعاملات گرافیکی، امکان Stream Video و... که این موارد حوزه هایی هستند که Javascript به لحاظ کارایی هنوز در آنها دچار مشکل است.

بازی های رایانه ای بخصوص بخشی است که علاوه براینکه نیاز به صدا و تصویر دارد، گاهی اوقات به فیزیک و هوش مصنوعی نیز وابسته است. جرقه افزایش کارایی به منظور اجرای بازی های رایانه ای در محیط وب می تواند باب جدیدی به منظور اجرای برنامه های مختلف در محیط وب باز کند و این کار قرار است با WebAssembly انجام شود.

۳.۱ چرا وب تا این حد جذاب است؟

زیبایی وب این است که مثل یک جادو است و در هرجا کار می کند. احتیاجی به نصب و دانلود ندارد و با یک کلیک به چیزی که نیاز داریم، پاسخ می دهد. این قضیه امنیت بیشتری نسبت به دانلود و اجرای کدهای باینری روی سیستم را فراهم می کند به دلیل آنکه مرورگرها دارای ملزومات امنیتی قابل قبولی برای نگهداری و اجرای کدها می باشند. همچنین اشتراک گذاری در وب به راحتی دریافت اطلاعات است. وب تنها پلتفرم جهانی است که امکان دسترسی برنامه ها در هر دستگاهی را فراهم می کند. تاکنون تعاملات وب اساسا توسط زبان Javascript انجام و پشتیبانی می شد که واقعا برای مقاصد جدیدی که به آنها اشاره شد طراحی نشده است.

۴.۱ WebAssembly چه چیزهایی به ارمغان می آورد؟

- سرعت (Speed)
- قابل حمل بودن (Portability)
- انعطاف پذیری (Flexibility)

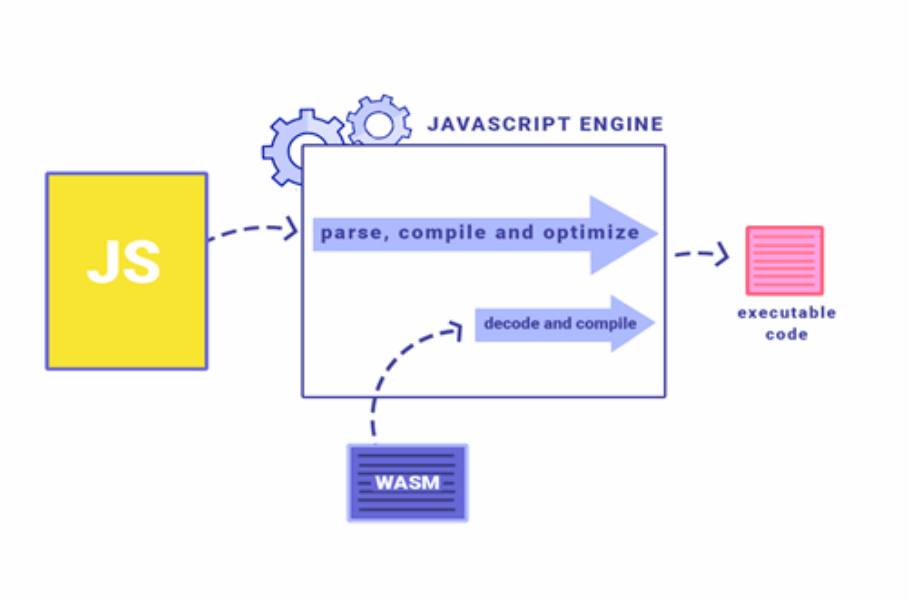
۵.۱ سرعت WebAssembly

WebAssembly برای افزایش سرعت طراحی ارائه شده است و شامل کدهای باینری است که حجم کمتری از فایل های متنی Javascript دارد و به علت حجم کم آن، سریعتر دانلود می شود و این فاکتور مهمی در شبکه های با سرعت پایین است. همچنین سریعتر رمزگشایی و اجرا می شود.

Javascript یک زبان با Type داینامیک است، یعنی نوع متغیر در ابتدا مشخص نمی شود که موضوع باعث افزایش سرعت و سهولت در کدنویسی می شود. اما به این معنی خواهد بود که موتور Javascript کار زیادی برای انجام دارد و نیاز است که کد Parse و کامپایل شود و برای اجرا در صفحات بهینه شود. عمل Parse کد Javascript، درگیر تغییر متن ساده به یک ساختار داده است که به آن Syntax (Abstract

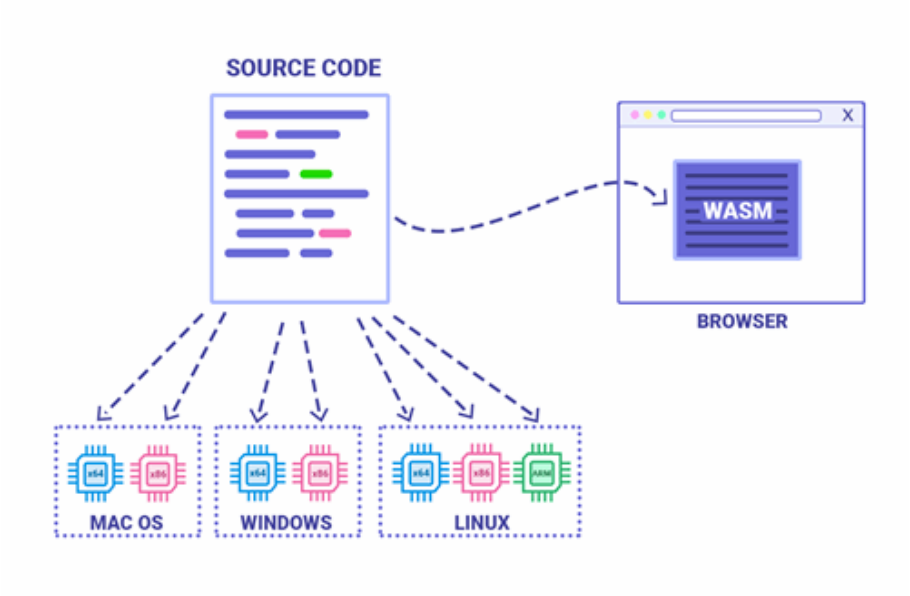
Tree(AST گفته می شود و متن را به فرمت باینری تبدیل می کند.

اما WebAssembly کد باینری ارائه می کند و رمزگشایی آن سریعتر انجام می شود. کدهای Native نوشته شده در WebAssembly بصورت Type ایستا هستند و در زمان کامپایل نیازی نیست که مشخص شود که نوع متغیر چیست، بنابراین اغلب بهینه سازی ها در هنگام کامپایل کد صورت می گیرد، قبل از آنکه به مرورگر ارائه شود. البته اجرای فایل های باینری WASM حدود ۲۰درصد کندتر از کد Native است.



۶.۱ قابل حمل بودن WebAssembly

یکی از اهداف WebAssembly قابل حمل بودن است. برای اجرای برنامه در یک دستگاه، آن برنامه باید با معماری پردازنده و سیستم عامل سازگار باشد. این قضیه به معنای آن است که کد باید به مجموعه ای از سیستم های عامل و هایCPU با معماری مختلف که می خواهیم پشتیبانی شود، کامپایل گردد. با WebAssembly یک مرحله کامپایل نیاز است و برنامه در هر مرورگر جدیدی اجرا می شود.



۷.۱ انعطاف پذیری WebAssembly

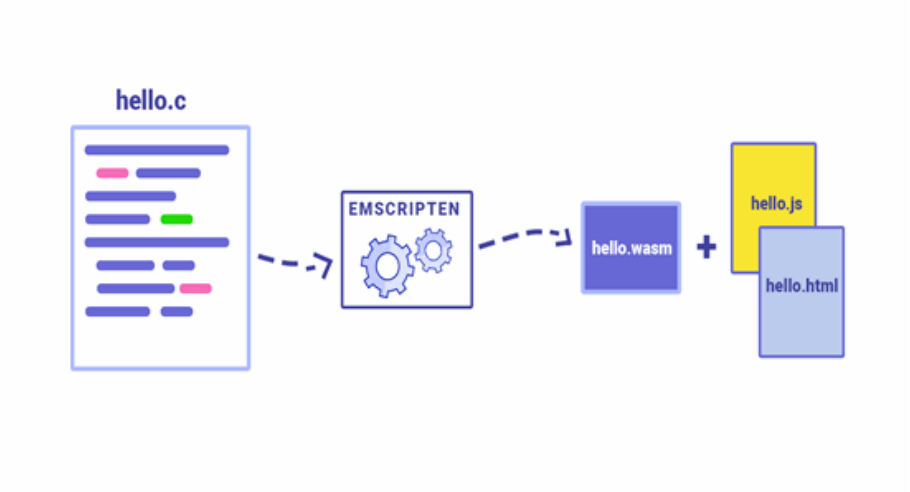
جالبترین مورد درباره WebAssembly انعطاف پذیری بیشتر آن در نوشتن برنامه های وب است. تاکنون مرورگرها فقط Javascript را پشتیبانی می کردند، اما با WebAssembly توسعه دهندگان می توانند از دیگر زبان ها برای توسعه و کدنویسی تحت وب استفاده کنند. Javascript هنوز هم بهترین انتخاب برای انجام بیشتر کارهاست و در صورتی که واقعا نیاز به بهبود کد نوشته شده باشد، اکنون گزینه های دیگری نیز وجود دارد. بنابراین می توان گلوگاه های موجود در کد را مجددا توسط زبان دیگری بازنویسی کرد.

۸.۱ WebAssembly چگونه کار می کند؟

به یک ابزار برای کامپایل کد نوشته شده به WebAssembly نیاز است. یکی از این ابزارها LLVM است که می تواند با زبان های مختلف کار کند. برای کامپایل C و C++ از یک کامپایلر مبتنی بر LLVM به نام Emscripten استفاده می شود. زبان Rust نیز دارای کامپایلر خودش به نام rustc است که می تواند مستقیما WebAssembly تولید کند.

اگر شما در زبان C کد " Hello World " را نوشته باشید ، می توانید با استفاده از خط فرمان Emscripten فایل های لازم برای اجرا در مرورگر را تولید کنید. این فایل ها شامل ماژول WebAssembly به همراه فایل های JS و HTML است.

```
emcc hello.c -s WASM=1 -o hello.html
```



به HTML و JavaScript نیاز داریم، چون WebAssembly نمی تواند مستقیما به هر پلتفرمی مثل WebAudio، WebGl، Dom و... دسترسی داشته باشد و برای کار با هریک از آنها باید کد WebAssembly هر صفحه توسط JavaScript فراخوانی شود. کامپایلر Emscripten کد JS موردنیاز برای استفاده ماژول WebAssembly را ایجاد می کند. HTML فایل JS را بارگذاری می کند و JS خروجی WebAssembly را در یک Textarea یا Canvas نمایش می دهد.

می توان اینطور تصور کرد که کدهای باینری WebAssembly درواقع ماژول های یک برنامه هستند که مرورگر می تواند آنها را بازیابی، بارگذاری و اجرا کند. کدهای WebAssembly دارای Import و Export هستند که می توان با آنها مثل اشیاء Javascript رفتار کرد. همچنین می توان توابع WebAssembly را در کد Javascript فراخوانی کرد و همینطور توابع Javascript را در WebAssembly فراخوانی کرد.

۹.۱ می توانیم از WebAssembly استفاده کنیم؟

بله! در حال حاضر چهار مرورگر اصلی یعنی Chrome Firefox، Edge و Safari از WebAssembly پشتیبانی می کنند و به مرور زمان WebAssembly در تمامی مرورگرهای موبایل و Desktop پشتیبانی خواهد شد.

جلسه ۲

Blazor Framework

روزبه غزوی - ۱۳۹۸/۵/۲۷

۱.۲ Blazor چیست؟

Blazor یک Framework برای ساختن وب اپلیکیشن های تک صفحه ای و یا اصطلاحاً Application Single-Page می باشد که با استفاده از آن می توانید کدهای سی شارپ و HTML را درکنار هم قرار بدهید تا در سمت Client و یا اصطلاحاً Client-Side اجرا بشوند. این نوع از برنامه ها بسیار شبیه به Application Razor Pages ها می باشند. کد نوشته شده به زبان سی شارپ به یک استاندارد به نام WebAssembly کامپایل خواهد شد که به شما اجازه می‌دهد از قابلیت‌های تمامی مرورگرهای وب امروزی استفاده کنید. در واقع با استفاده کردن از Blazor Framework به سادگی می توانیم اپلیکیشن های Client-Side که از سی شارپ و .NET Core استفاده می کنند را ایجاد کنیم. استفاده کردن از Blazor Framework برای تمامی دوستانی که امکان استفاده کردن از زبان JavaScript برای ساختن برنامه های Client-Side را ندارند بسیار مناسب می باشد؛ به عبارت دیگر با استفاده کردن از Blazor شما دیگر نیازی به استفاده کردن از زبان JavaScript برای نوشتن کدهای سمت Client نیستید و به راحتی می توانید با استفاده از سی شارپ و همچنین .NET Core برنامه هایی بنویسید که کدهای سی شارپ را در سمت Client و در مرورگر کاربر اجرا می کنند.

مزایای استفاده از فریم ورک Blazor:

- ایجاد رابط کاربری قوی و تعاملی توسط سی شارپ بجای استفاده از JavaScript
- اشتراک منطق برنامه نویسی client-side و server-side توسط NET.
- رندر رابط کاربری شامل HTML و CSS با پشتیبانی از گستره زیادی از مرورگرها شامل مرورگرهای تلفن همراه

استفاده از NET. برای توسعه دهنده سمت کاربر مزایای زیر را شامل می شود:

- کدنویسی سی شارپ بجای JavaScript
- استفاده از کتابخانه های متعدد اکوسیستم NET.
- اشتراک منطق برنامه سمت server و client
- ایجاد یک مجموعه مشترک از زبان ها، چارچوب ها و ابزار که پایداری، سهولت و ویژگی های غنی را به همراه خواهد داشت.

۲.۲ کامپوننت ها

برنامه تولید شده توسط Blazor مبتنی بر کامپوننت است. یک کامپوننت در Blazor یک عنصر رابط کاربری می باشد، مثل یک صفحه یا باکس گفتگو و یا فرم ورود اطاعات. کامپوننت ها رویدادهای کاربر را مدیریت می کنند و میزان انعطاف رابط کاربری هنگام رندر را تعیین می کنند. کامپوننت ها بصورت کلاس در اکوسیستم NET. ایجاد می شوند. همچنین کامپوننت ها معمولاً توسط صفحات Razor نوشته می شوند.

گاهی اوقات کامپوننت ها همان کامپوننت های صفحات Razor هستند. Razor یک syntax برای ترکیب نشانه های HTML و کد سی شارپ می باشد. MVC و Razor Page از Razor Syntax استفاده می کنند. برخلاف MVC و Razor Page که از مدل درخواست و پاسخ استفاده می کنند، کامپوننت ها به طور خاص فقط برای منطق رابط کاربری استفاده می شود.

کد زیر یک کامپوننت Razor ایجاد می کند که خود این کامپوننت می تواند در کامپوننت دیگری نیز استفاده شود.

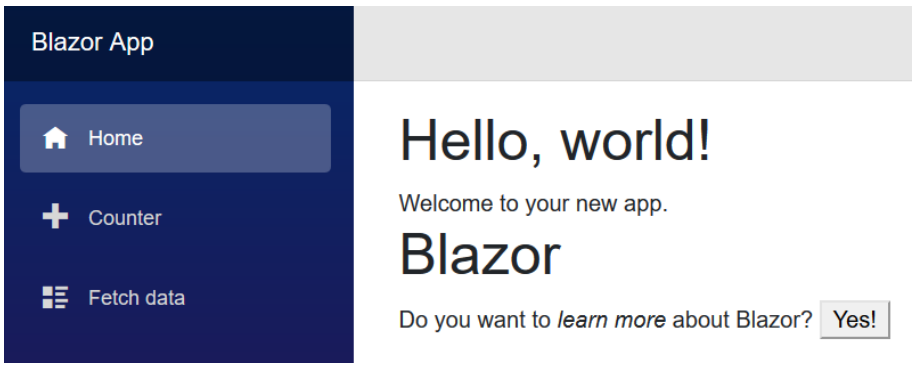
```
<div>
    <h1>@Title</h1>

    @ChildContent

    <button onclick="@OnYes">Yes!</button>
</div>

@functions {
    [Parameter]
    private string Title { get; set; }

    [Parameter]
    private RenderFragment ChildContent { get; set; }

    private void OnYes()
    {
        Console.WriteLine("Write to the console in C#!");
    }
}
```

در این قطعه کد (که بیان کننده کامپوننت یک Dialog است) از کامپوننت ها ChildContent و Title که قبلاً ایجاد گردیده، استفاده شده است. همچنین OnYes یک متد سی شارپ است که رویداد onclick را راه اندازی می کند.

Blazor از tag های متداول HTML برای ایجاد رابط کاربری استفاده می کند. عناصر HTML کامپوننت ها را مشخص می کنند و ویژگی های تگ ها مقادیر را به پارامترهای موجود در کامپوننت ها پاس می دهند. ChildContent و Title پارامترهای کامپوننت Dialog هستند و توسط کامپوننت دیگری که از کامپوننت Dialog استفاده می کند، مقداردهی می شوند.
در مثال زیر کامپوننت Dialog توسط Index.razor استفاده شده است.

```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.

<Dialog Title="Blazor">
    Do you want to <i>learn more</i> about Blazor?
</Dialog>
```

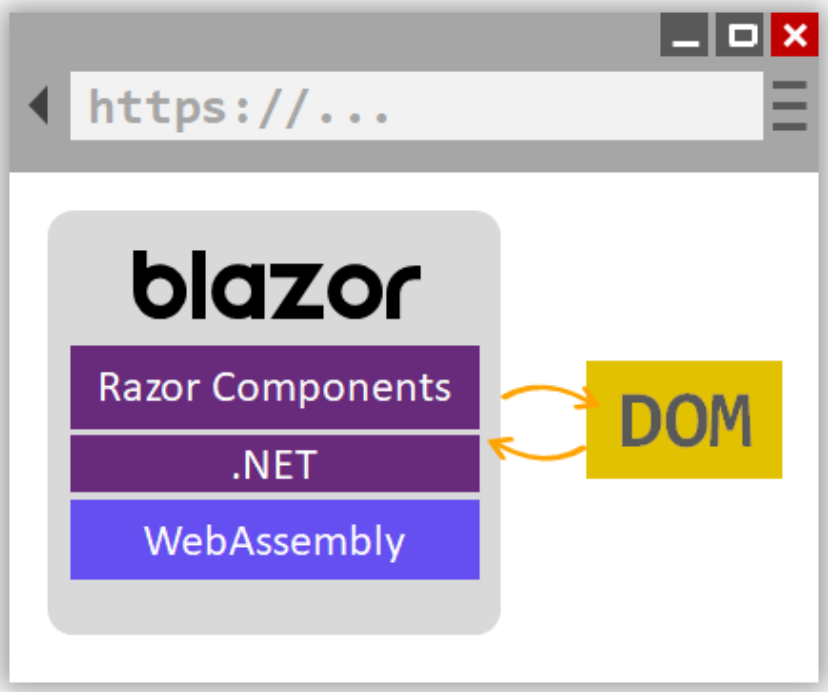
هنگامی که Index.razor توسط مرورگر بارگزاری می شود، کامپوننت Dialog رندر می شود.



Blazor Client-Side ۳.۲

Blazor سمت کاربر یک برنامه با چارچوب کاری تک صفحه ای است که باعث ایجاد برنامه های تعاملی توسط فرم ورک .NET می شود. Blazor سمت کاربر از استانداردهای متداول وب استفاده می کند که بدون هیچ پلاگین و کد رمزگذاری شده ای در تمامی مرورگرهای مدرن شامل مروگرهای موبایل کار می کند. اجرای کد فریم ورک .NET در مرورگر توسط WebAssembly بطور خلاصه wasm امکان پذیر می شود. WebAssembly یک استاندارد وب است و بدون نیاز به پلاگین توسط مرورگرهای وب پشتیبانی می شود. WebAssembly در فرمت فشرده بایت کد است که جهت افزایش سرعت دانلود و افزایش حداکثری سرعت اجرا، بهینه شده است.

کد WebAssembly توسط JavaScript می تواند به تمامی عملکردها یا functionality مرورگر دسترسی داشته باشد که به آن Javascript Interoperability گفته می شود. کد .NET توسط WebAssembly در مرورگر اجرا می شود، در همان Sandbox مطمئنی که JavaScript اجرا می شود و که عملاً فرصت انجام عملیات مخرب توسط برنامه روی دستگاه کاربر را از بین می برد.



هنگامی که یک برنامه سمت کاربر ایجاد شده توسط Blazor در مرورگر اجرا می شود:

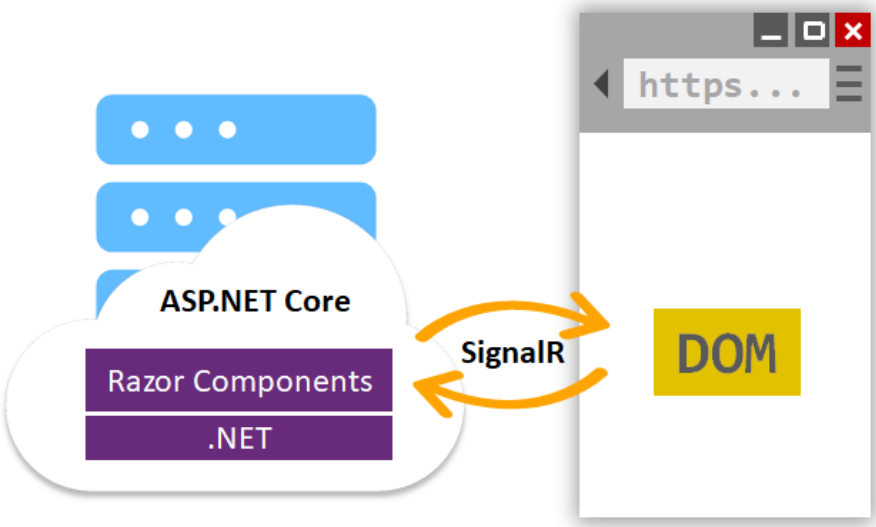
- فایل کدهای سی شارپ و فایل های Razor به .NET کامپایل می شوند.
- .NET Runtime برای مرورگر بارگذاری می شود.
- کدهای Bootstrap و تنظیمات و .NET Runtime برای مجموعه نرم افزار بارگذاری می شود Blazor Runtime سمت کاربر از Javascript Interoperability برای مدیریت و تغییر DOM و فراخوانی API ها استفاده می کند.

اندازه برنامه منتشر شده را Payload Size می گویند. Payload Size فاکتور خیلی مهم در مورد قابل استفاده بودن یک برنامه است. یک برنامه حجیم زمان نسبتاً طولانی جهت دانلود در مرورگر نیاز دارد که به تجربه کاربری لطمه وارد می کند. در Blazor سمت کاربر Payload Size بهینه شده تا زمان دانلود را کاهش داده شود:

- کد های بلا استفاده یعنی همان Intermediate Language یا به اختصار IL توسط پیوند دهنده یا همان Linker حذف میشوند.
- درخواست های HTTP فشرده سازی می شوند.
- مجموعه .NET و Runtime آن در مرورگر کش می شوند.

۴.۲ Blazor Server-Side

Blazor منطق رندر کردن کامپوننت ها را از چگونگی بروزرسانی رابط کاربری، مجزا می کند. Blazor سمت سرور توسط برنامه ASP.NET Core پشتیبانی و میزبانی می شود. همچنین بروزرسانی های رابط کاربری توسط ارتباط SignalR مدیریت می شود. مدیریت کننده های Runtime رویدادها را از مرورگر به سرور ارسال می کنند و سرور بروزرسانی های مورد نیاز را بعد از اجرای کامپوننت ها به مرورگر ارسال می کند. ارتباط استفاده شده توسط Blazor سمت سرور برای تعامل با مرورگر، برای فراخوانی های مورد نیاز JavaScript Interoperability نیز استفاده می شود.



۵.۲ JavaScript Interoperability

برای برنامه هایی که نیاز به کتابخانه های JavaScript و API های مرورگر دارند، کامپوننت ها با JavaScript تعامل می کنند. کامپوننت ها می توانند هر کتابخانه و یا API ای که Javascript از آن استفاده می کند را بکار بگیرند. کد سی شارپ می تواند داخل کد JavaScript فراخوانی شود و همچنین کد JavaScript می تواند داخل کد سی شارپ فراخوانی شود.

۶.۲ اشتراک کد و .NET. استاندارد

برنامه ها می توانند از کتابخانه های .NET. استاندارد استفاده کنند. .NET. استاندارد خصوصیات رسمیِ API های .NET. است که در تمامی نوع های متداول .NET. پیاده سازی شده است. Blazor پیاده سازی شده با استاندارد API های .NET. در مرورگرهای وب کاربرد ندارد برای مثال در دستیابی به فایل سیستم، بازکردن یک سوکت و Threading استفاده می شود. کتابخانه های استاندارد .NET. می توانند در پلتفرم های مختلف .NET. اشتراک گذاشته شوند مثل Blazor و .NET Framework. و .NET Core. و Xamarin و Mono و Unity

۷.۲ تفاوت Blazor Server و Blazor WebAssembly

WebAssembly Hosting Model

- WASM runs in the browser on the client.
- The first request to the WASM application downloads the CLR, Assemblies, JavaScript, CSS (React and Angular work similar).
- It runs in the secure WASM sandbox.
- The Blazor Javascript handler accesses the DOM (Document Object Model).

Server Hosting Model

- The C# code runs on the server.
- Javascript hooks are used to access the DOM.
- Binary messages are used to pass information between the browser and the server using SignalR.
- If something is changed the server sends back DOM update messages.

۸.۲ آشنایی بیشتر

برای آشنایی بیشتر با فریمورک بلیزر و همچنین چگونگی کارکرد آن می توانید به لینک زیر مراجعه کنید.
https://www.aparat.com/Hamcker/videos

جلسه ۳

Getting Started With Blazor

روزبه غزوی - ۱۳۹۸/۶/۷

در این بخش به نصب و راه اندازی فریم ورک Blazor و همچنین بررسی یک پروژه نمونه (Demo) از آن می پردازیم.

۱.۳ پیشنیاز ها

۱.برای شروع به کار ابتدا نیاز به یک IDE یا کد ادیتور مناسب داریم. IDE پیشنهادی مایکروسافت برای کار با فریم ورک Blazor برنامهٔ Visual Studio ۲۰۱۹ می باشد اما میتوان از IDE سبک تر مایکروسافت یعنی Visual Studio Code به همراه افزونه سی شارپ نیز بهره برد.

دانلود Visual Studio ۲۰۱۹
دانلود Visual Studio Code

برای آشنایی با IDE و مفهوم آن میتوانید به لینک روبرو مراجعه کنید: IDE چیست

۲.پس از نصب و راه اندازی IDE مورد نظر خود باید Blazor را از آدرس Blazor.net دانلود و نصب کنید تا Template های مربوط به WebAssembly ایجاد شده و در برنامه Visual Studio قابل استفاده باشند.
۳.پس از نصب .NET SDK. برای اطمینان از کارکرد صحیح آن یک پنجره جدید CMD باز کرده و در آن دستور زیر را تایپ کنید.

```
dotnet
```

اگر پس از اجرای دستور فوق اطلاعاتی در مورد چگونگی کار با Dotnet به نمایش درآمد برنامه به درستی نصب شده است.
توجه کنید که برای استفاده از Blazor WebAssembly باید حتما از نسخه ۳.۱ .NET Core SDK یا جدید تر استفاده کنید.

۲.۳ ساخت اولین پروژه Blazor

برای ساخت پروژه Blazor ابتدا یک پوشه دلخواه را انتخاب کرده و سپس یک پنچره CMD در آن باز کنید.
برای اینکار می توانید به پوشه موردنظر خود رفته سپس از منوی File در بالای صفحه گزینهٔ
Open Windows Powershell را انتخاب نمایید.

پس از باز کردن CMD یا PowerShell در پوشه مورد نظر، برای ساختن پروژه WebAssembly دستور زیر را اجرا کنید.

```
dotnet new blazorwasm --hosted --output BlazorDemoApp
```

BlazorDemoApp نام پروژه شماست و می توانید به دلخواه آن را تغییر دهید. همچنین برای آشنایی بیشتر با تنظیمات پروژه می توانید از دستور زیر استفاده کنید.

```
dotnet new blazorwasm --help
```

برای ساخت پروژه Blazor Server می توانید از دستور زیر استفاده کنید. اما هدف ما از این بخش ساخت پروژه Blazor WebAssembly است.

```
dotnet new blazorserver -o BlazorDemoApp --no-https
```

پس از ساخت پروژه Blazor نوبت به راه اندازی آن می رسد. برای راه اندازی ابتدا باید به پوشه پروژه ساخته شده رفت و سپس از پوشه Server یا Client آن را راه اندازی کرد. برای این کار دستورات زیر را به ترتیب در CMD یا PowerShell اجرا نمایید.

```
cd BlazorDemoApp
cd Server
dotnet run
```

اگر همه چیز درست پیش رفته باشد پس از ران کردن پروژه، اطلاعات زیر نمایش داده خواهد شد:

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path:BlazorDemoApp/Server
```

برای مشاهده پروژهٔ خود یک مرورگر دلخواه باز کرده (ترجیحاً Chrome یا FireFox) سپس یکی از آدرس های نمایش داده شده در CMD .را در آن وارد کنید

```
https://localhost:5001
http://localhost:5000
```

نکته ۱: ممکن است پورت های ۵۰۰۰ یا ۵۰۰۱ در کامپیوتر شما درگیر بوده و پروژه روی پورت دیگری اجرا شود پس برای مشاهده پروژه، آدرس نمایش داده شده در CMD .خود را وارد کنید

نکته ۲: ممکن است هنگام وارد کردن آدرس سرور با ارور Connection Is Not Secure مواجه شوید که برای رفع این مشکل کافیست آدرس را به لیست Exception ها اضافه کرده و امنیت آن را تضمین کنید.

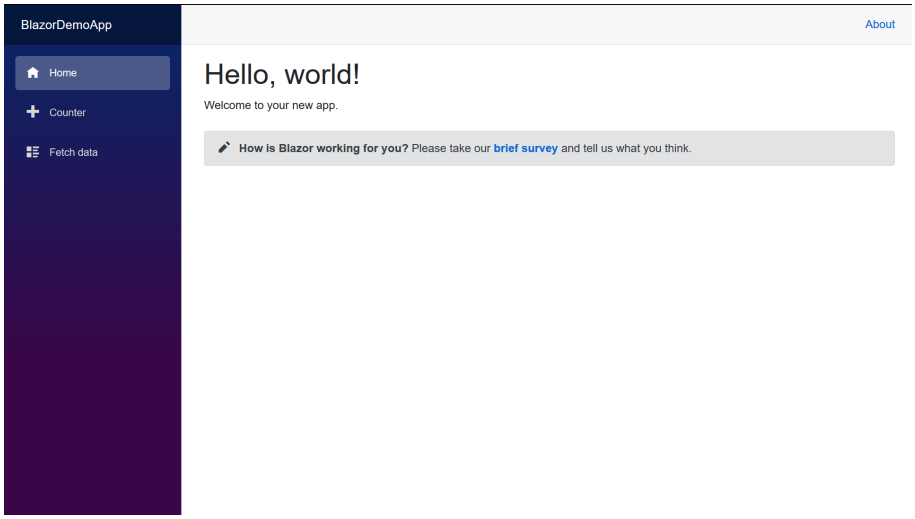
اگر تمام مراحل فوق به درستی انجام شده باشد پس از وارد کردن آدرس در مرورگر، اپلیکیشن Demo نمایش داده خواهد شد.

- اپلیکیشن دموی بلیزر شامل ۳ صفحه ی Home و Counter و Fetch Data می باشد.
- این صفحات توسط سه فایل Razor در پوشه Pages پیاده سازی شده اند.
- هر کدام از این فایل ها یک مؤلفه Blazor را پیاده سازی میکند که در مرورگر، سمت کاربر کامپایل و اجرا می شود.

۳.۳ Home Page

پیاده سازی صفحه Home در پوشه Pages و در فایل Index.razor صورت گرفته است.

```
۱  @page "/"
۲
۳  <h1>Hello, world!</h1>
۴
۵  Welcome to your new app.
۶
۷  <SurveyPrompt Title=" How is Blazor working for you? " />
```



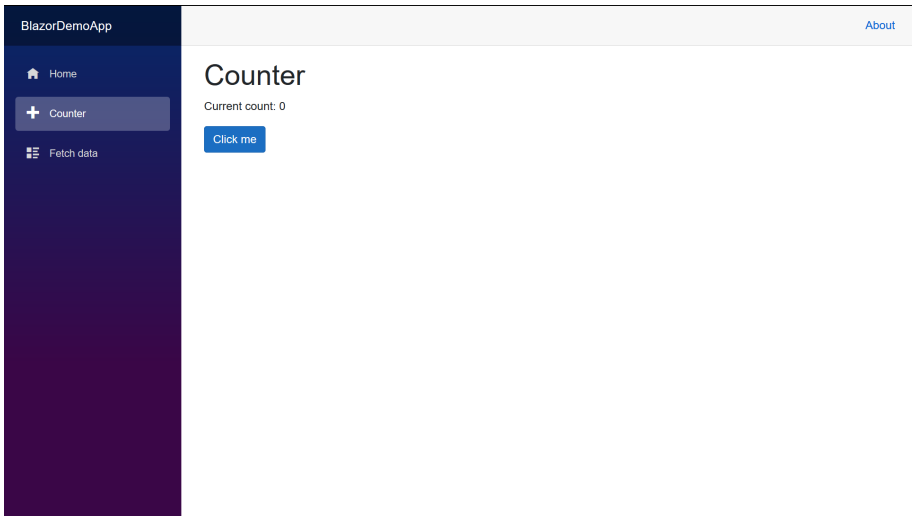
۴.۳ Counter Page

صفحه شمارنده دارای یک دکمه می باشد که با هر بار انتخاب دکمه، شمارنده بدون refresh صفحه افزایش می یابد.

پیاده سازی صفحهٔ شمارنده در فایل Counter.razor قرار دارد.

```
@page " /counter "

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class=" btn btn-primary " @onclick=" IncrementCount ">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

به طور معمول، این نوع رفتار سمت کاربر توسط جاوا اسکریپت (JavaScript) اداره می شود، اما در اینجا با استفاده از کامپوننت شمارنده (Counter) با سی شارپ و .NET پیاده سازی شده است.

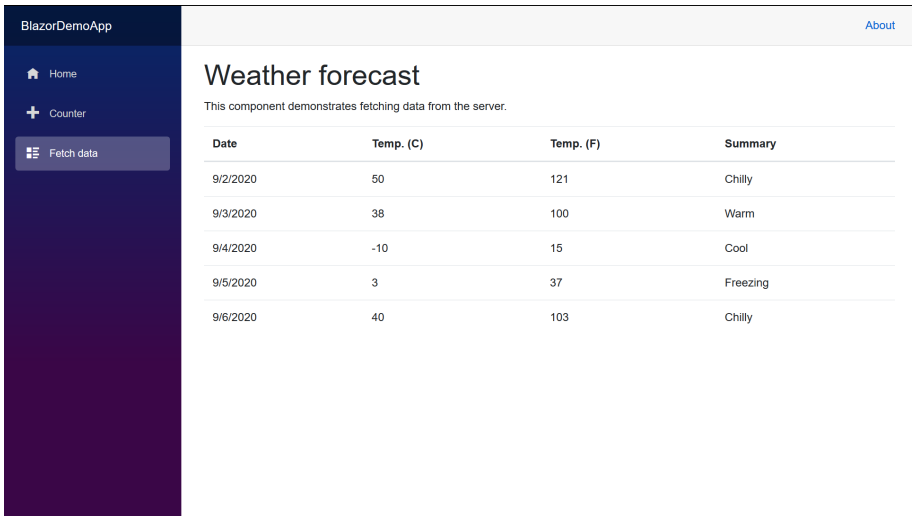


۵.۳ Fetch Data Page

نگاهی به پیاده سازی کامپوننت دسترسی به داده در فایل FetchData.razor بیاندازید.

```
@page "/fetchdata"
@using BlazorDemoApp.Shared
@inject HttpClient Http

<h1>Weather forecast</h1>

<p>This component demonstrates fetching data from the server.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temp. (C)</th>
                <th>Temp. (F)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var forecast in forecasts)
            {
                <tr>
                    <td>@forecast.Date.ToShortDateString()</td>
                    <td>@forecast.TemperatureC</td>
                    <td>@forecast.TemperatureF</td>
                    <td>@forecast.Summary</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
    private WeatherForecast[] forecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("Weather");
    }

}
```

از بخش @inject برای تزریق یک نمونه HttpClient به درون کامپوننت استفاده می شود. کامپوننت FetchData برای بازیابی داده JSON از سرور در زمان اولیه سازی (initialize) کامپوننت، از HttpClient تزریق شده استفاده می کند.



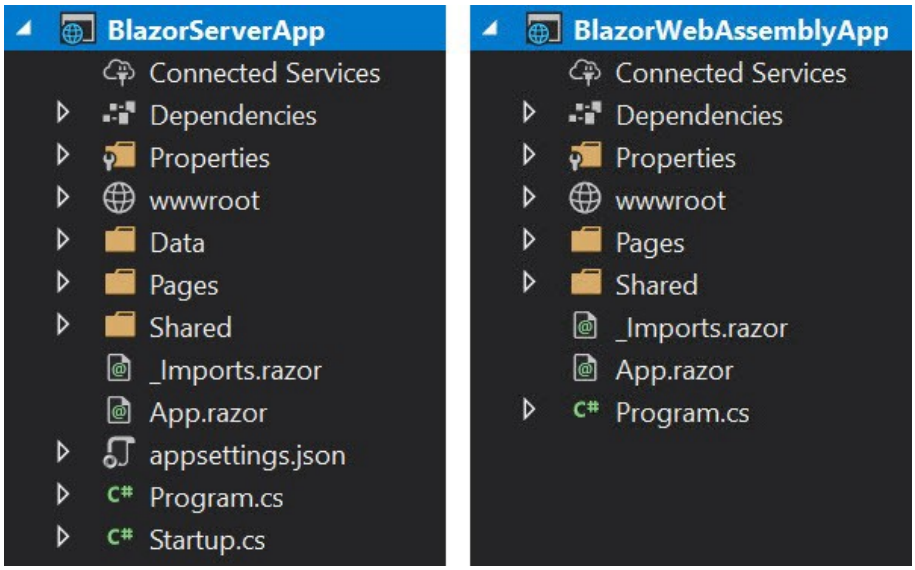
۶.۳ Blazor Project Structure

در ابتدا فولدرهای Dependencies و Properties و wwwroot را داریم که شبیه به یک اپلیکیشن معمولی ASP.NET Core می باشند.

- فولدر Pages شامل صفحاتی است که نهایتاً برنامه را تشکیل می دهند که شبیه به یک اپلیکیشن Page Razor عمل می کند.
- فولدر Shared حاوی فایل های Layout می باشد که در سرتاسر برنامه مورد استفاده قرار می گیرند.
- فایل Imports.razor برای Import کردن Namespace ها به درون دیگر فایل های razor مورد استفاده قرار می گیرد.
- فایل Program.cs برای ساختن Hosting Environment مربوط به ASP.NET Core مورد استفاده قرار می گیرد.
- فایل Startup.cs برای ساختن تنظیمات مربوط به پروژه Blazor و Dependency های آن مورد استفاده قرار می گیرد.

- نهایتاً فایل App.razor شامل تنها فایلی است که مربوط به برنامه Blazor می باشد.

[]



برای آشنایی بهتر با ساختار پروژه و چگونگی کارکرد هر فایل می توانید به لینک زیر مراجعه کنید:

Blazor Project Structure

نکته:ساختار پروژه بلیزر ممکن است در مرور زمان تغییر کند و این ساختار مربوط به تاریخ نوشته شدن این مطلب می باشد پس برای اطلاع یافتن از آخرین تغییرات بهتر است به داک مایکروسافت مراجعه کنید.

جلسه ۴

First Progressive Webapp

روزبه غزوی - ۱۳۹۸/۶/۱۲

۱.۴ PWA چیست؟

PWA نوعی اپلیکیشن تک صفحه ای (SPA) است که با استفاده از امکانات مرورگر های مدرن، شبیه به اپلیکیشن های دسکتاپ عمل می کند. با استفاده از Blazor WebAssembly که نسخه Candidate Release آن اخیرا منتشر شده، می توانید اپلیکیشن های PWA ایجاد کنید. در این بخش، اولین برنامه PWA را ایجاد می کنیم. سورس پروژه در بخش منابع در دسترس است. در این قسمت به صورت خلاصه به آموزش PWA در Blazor می پردازیم.

برای آشنایی بیشتر با مفهوم وب اپلیکیشن پیش رونده یا به اختصار (PWA) به این لینک مراجعه کنید.

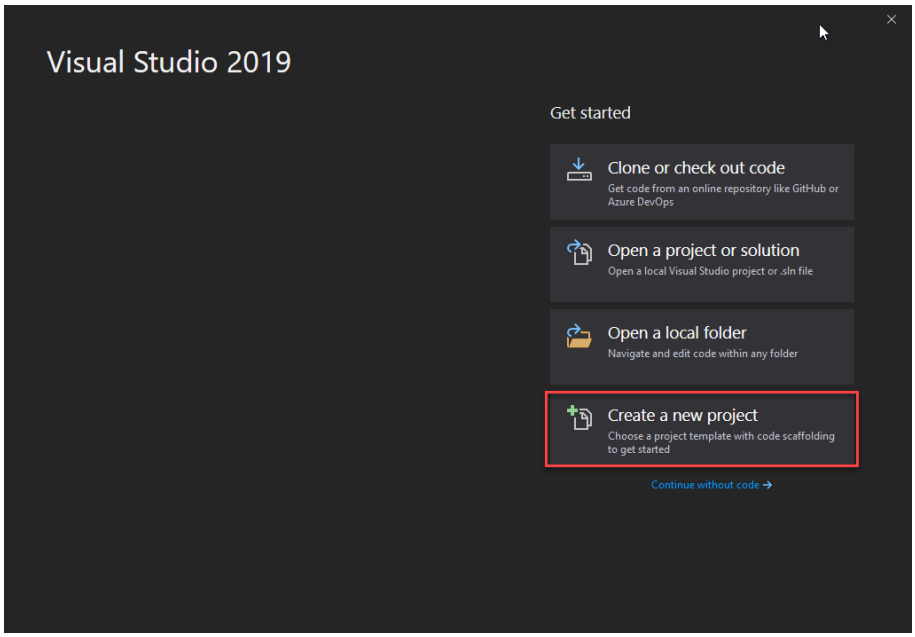
۲.۴ ساخت یک پروژه PWA با Blazor

اگر از برنامه Visual Studio Code به عنوان IDE خود استفاده می کنید کافیست همانند بخش قبل در یک پوشهٔ دلخواه دستور زیر را در CMD یا Powershell وارد کرده و سپس پوشه پروژه ساخته شده را با VSCode باز کنید.

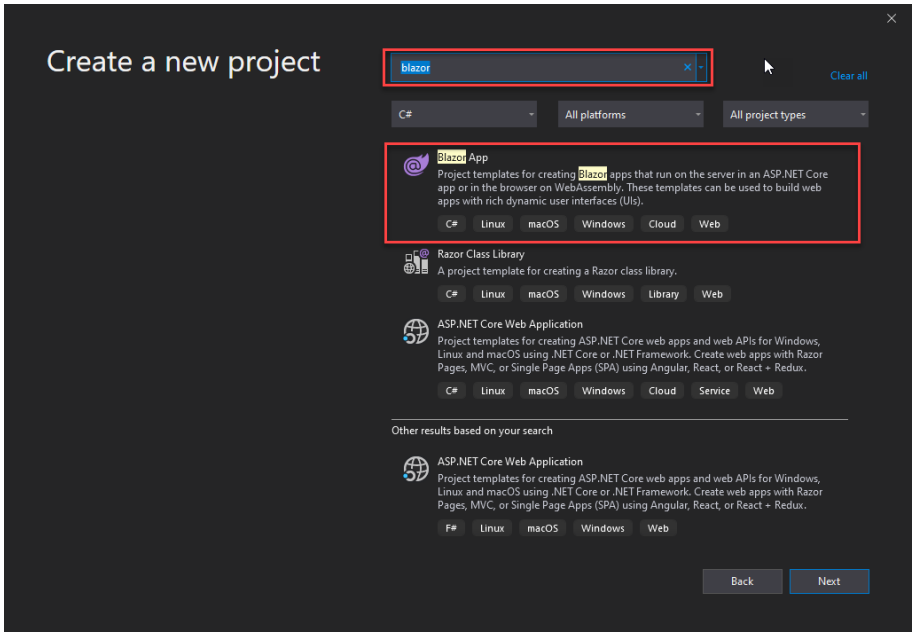
```
dotnet new blazorwasm --hosted --output BlazorPwaTodo
```

*اگر از Visual Studio استفاده می کنید، مراحل زیر را دنبال کنید:

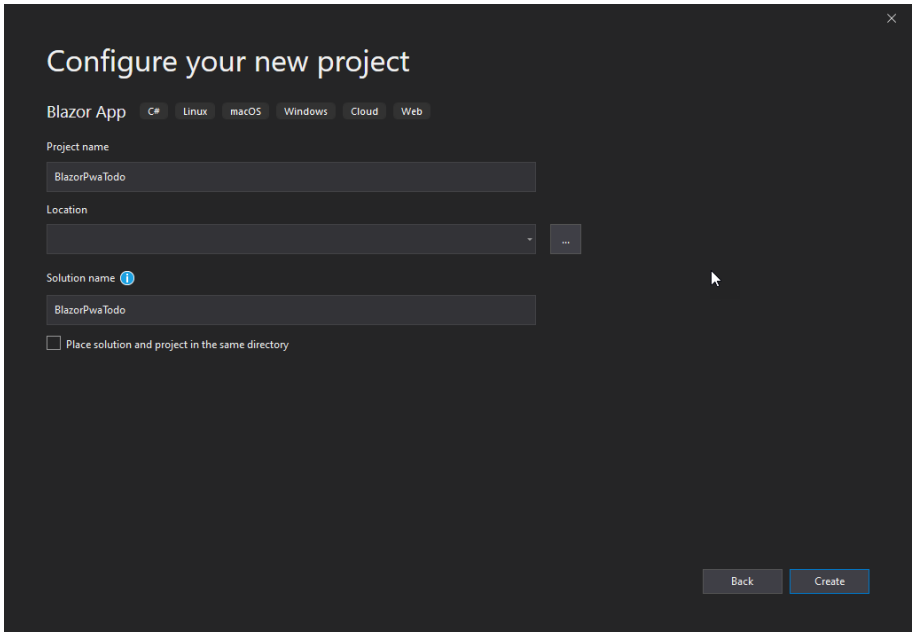
بعد از اجرای Visual Studio روی گزینه Create A New Project کلیک کنید:



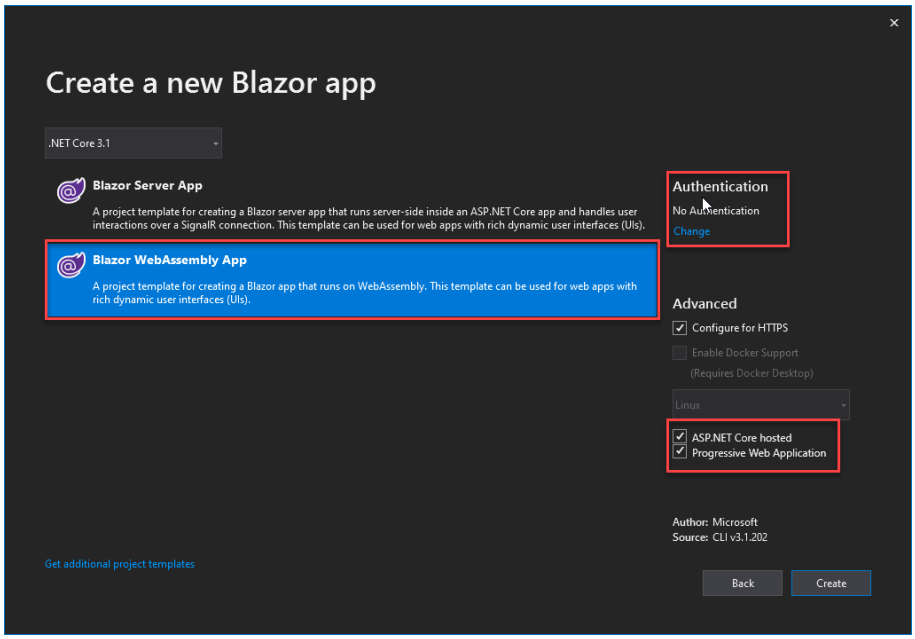
در صفحه جدید می توانید با سرچ واژه Blazor پروژه جدید از نوع Blazor App ایجاد کنید:



سپس نام و آدرس محل دخیره سازی پروژه را وارد می کنیم:



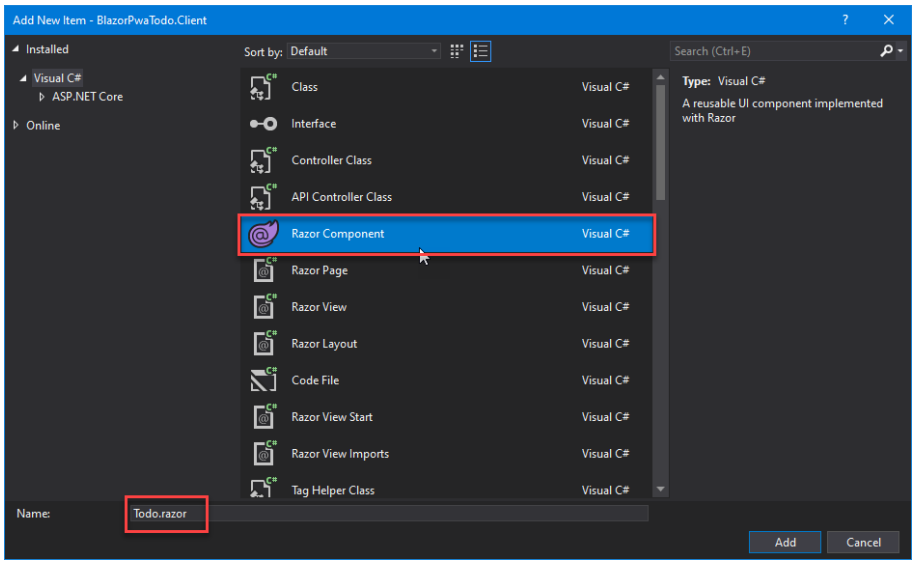
در ادامه همانند تصویر زیر گزینه های مورد نظر را انتخاب کنید. گزینه ASP.NET Core Hosted به این دلیل انتخاب شده که در بخش های بعدی مورد استفاده قرار خواهد گرفت تا امکانات بیشتری اضافه شود. به علاوه اجرای پروژه در محیط توسعه آسان تر می شود.



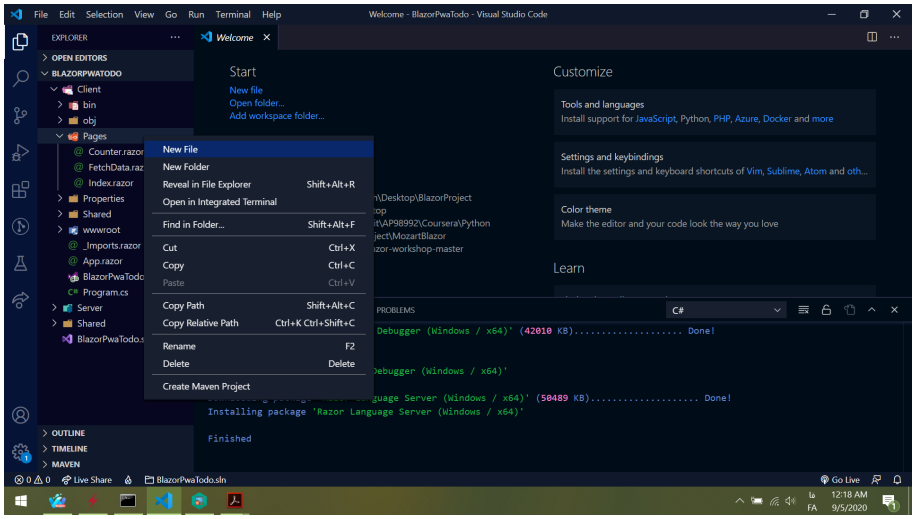
در آخر روی دکمه Create کلیک کرده تا پروژه ساخته شود.

۳.۴ ساختن و افزودن یک Component

در این پروژه PWA با Blazor میخواهیم یک اپ ToDo ایجاد کنیم پس اولین کامپوننت را در پوشه Pages با نام Todo ایجاد می کنیم. برای این کار روی Pages کلیک راست کنید و مسیر Add > New Item > Razor Component دنبال کنید.

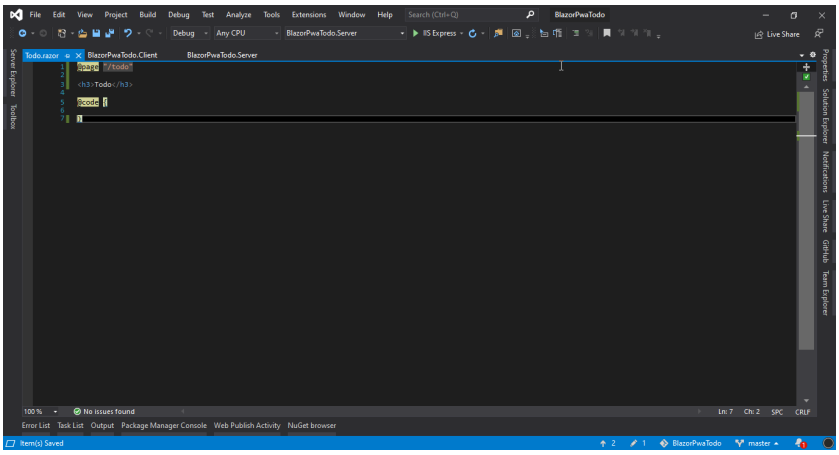


در VSCode نیز به همین ترتیب روی پوشه Pages کلیک راست کرده سپس گزینه New File را انتخاب کرده و فایل Todo.razor را ایجاد کنید.

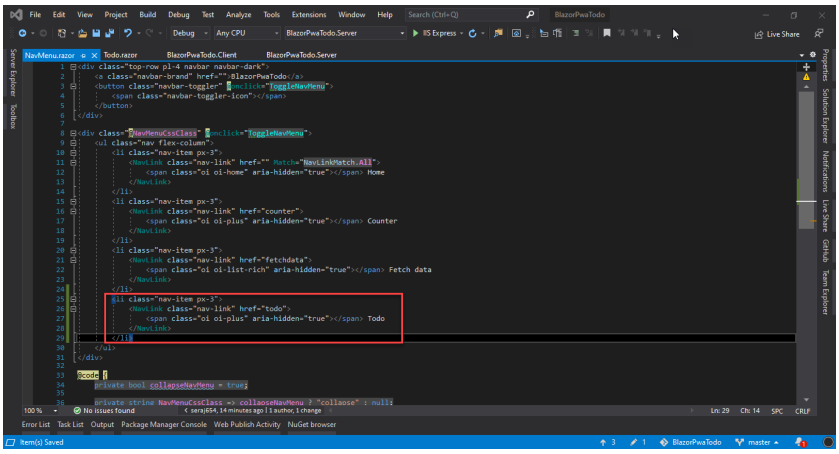


سپس طبق تصویر، کد زیر را در آن جایگذاری کنید. خط اول کد، مشخص کننده Route این کامپوننت است. پس با ورود به todo/ در مرورگر، این کامپوننت نمایش داده می شود.

```
@page "/todo"

<h3>Todo</h3>

@code{

}
```



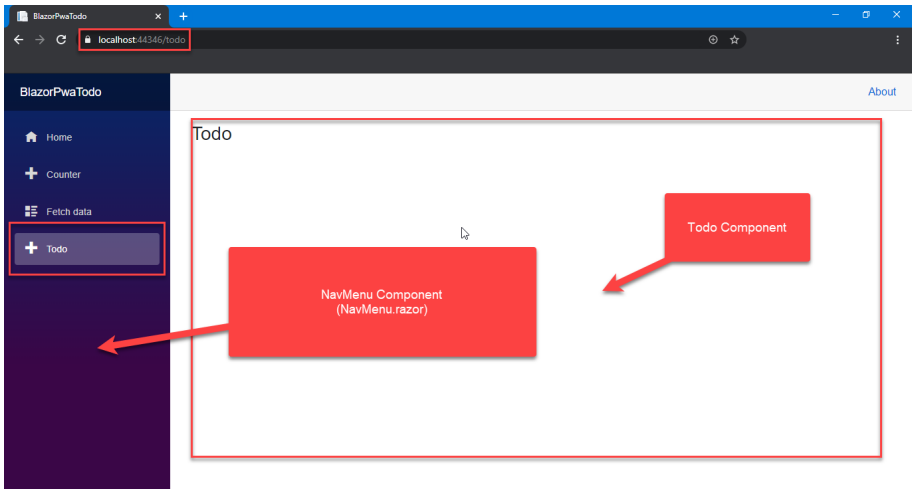
سپس می توانید با ویرایش فایل NavMenu.razor که در پوشه Shared قرار دارد، این کامپوننت را به منو سایت (اپلیکیشن) اضافه کنید.



برای اینکار به پوشه Client/Shared رفته و کد زیر را مانند عکس قبل در فایل NavMenu.razor اضافه کنید.

```
<li class=" nav-item px-3 ">
    <NavLink class="nav-link" href="todo">
        <span class=" oi oi-list-rich " aria-hidden="true"></span> Todo
    </NavLink>
</li>
```

حال، می توانید پروژه را اجرا کنید. خروجی مانند تصویر زیر است:

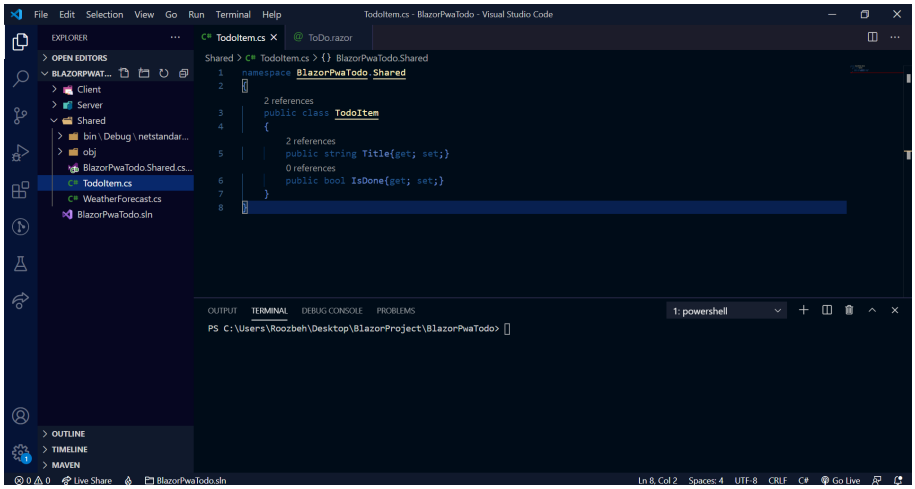


در VSCode می توانید از ترمینال درون برنامه استفاده کنید یا یک پنجره جدید CMD باز کرده و دستورات زیر را در آن اجرا کنید.

```
cd BlazorPwaTodo
cd Server
dotnet run
```

۴.۴ ساخت امکانات اولیه Todo

همانطورکه میدانید، پس از ساخت پروژه از نوع Blazor WebAssembly در solution سه پروژه داریم. در پروژه Shared کلاس TodoItem را ایجاد می کنیم:

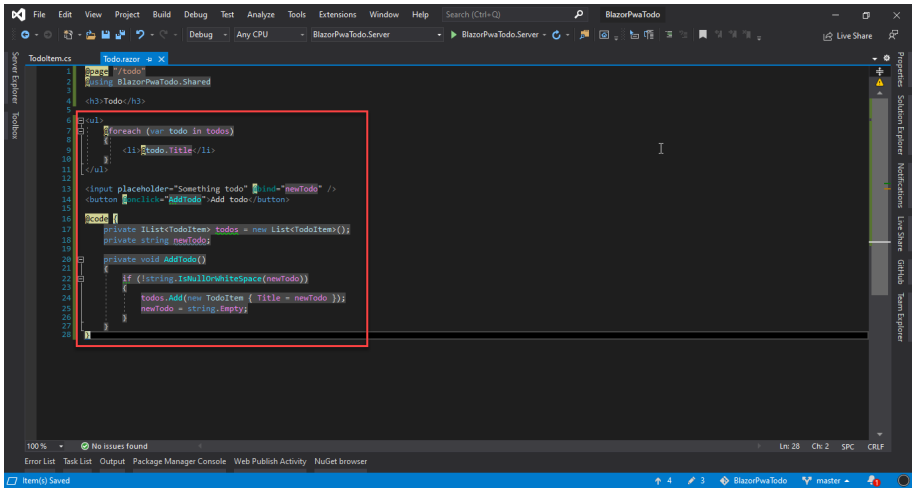


پس از ایجاد کردن کلاس TodoItem.cs کد زیر را در آن وارد کنید:

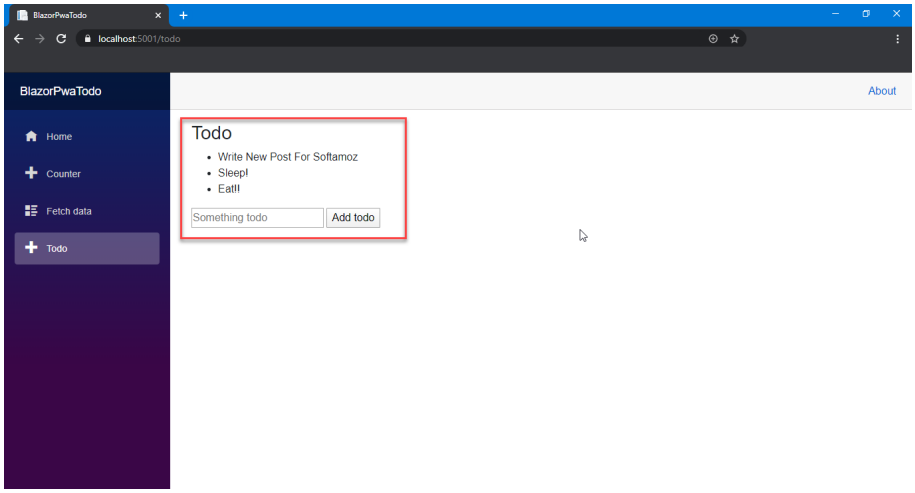
```
۱  namespace BlazorPwaTodo.Shared
۲  {
۳      public class TodoItem
۴      {
۵          public string Title{get; set;}
۶          public bool IsDone{get; set;}
۷      }
۸  }
```

سپس به پوشه Pages برگشته و در کامپوننت ToDo.razor موارد زیر را اضافه کنید:

```
@page "/todo"
@using BlazorPwaTodo.Shared

<h3>Todo</h3>

<ul>
    @foreach (var todo in todos)
    {
        <li>@todo.Title</li>
    }
</ul>

<input placeholder=" Something todo " @bind="newTodo" />
<button @onclick="AddTodo">Add todo</button>

@code
{
    private IList<TodoItem> todos =new List<TodoItem>();
    private string newTodo;
    private void AddTodo()
    {
        if(!string.IsNullOrWhiteSpace(newTodo))
        {
            todos.Add(new TodoItem{ Title=newTodo });
            newTodo=string.Empty;
        }
    }
}
```

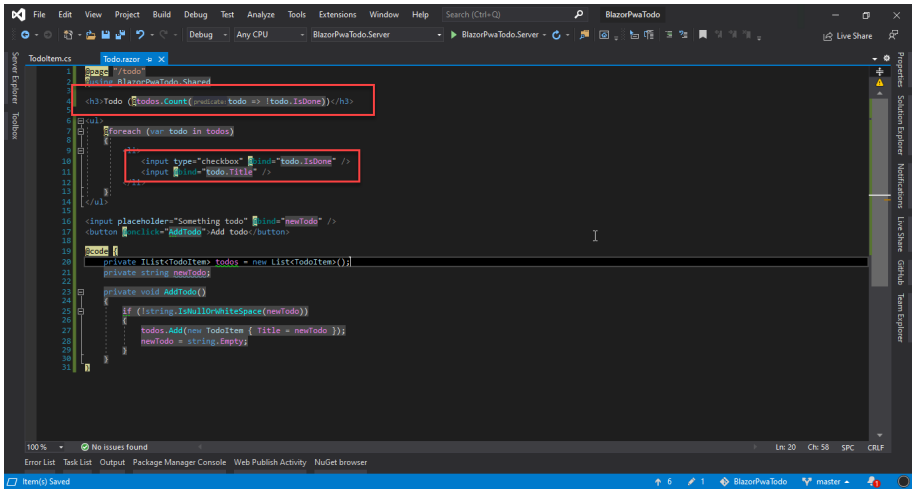


در قسمت Code لیستی از TodoItem ها داریم که با استفاده از foreach آیتم های آن را نمایش می دهیم. با استفاده از AddTodo و input می توانیم todo های جدید اضافه کنیم. در صورتی که با این کد ها آشنایی ندارید، می توانید در مورد Blazor تحقیق کنید. سپس پروژه را اجرا میکنیم. خروجی به شکل زیر است:



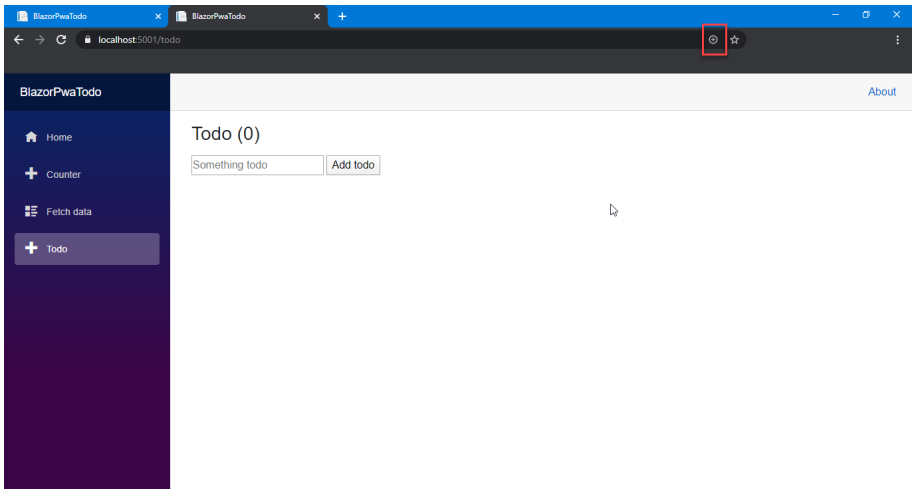
۵.۴ علامت زدن Todo به عنوان انجام شده

همانند تصویر زیر، تغییراتی در کامپوننت Todo انجام می دهیم تا بتوانیم، Todo ها انجام شده را تیک بزنیم:

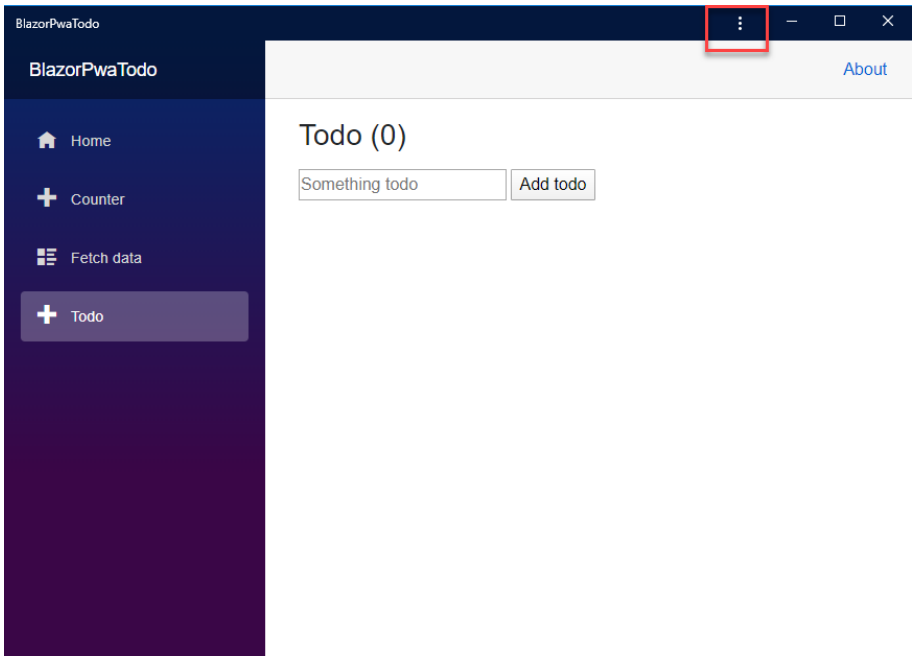


۶.۴ نصب PWA

تا به حال یک برنامه ساده ساخته ایم. با هربار اجرا امکان استفاده از PWA وجود دارد. پس می توانیم برنامه برا به صورت آفلاین اجرا کنیم، حتی آن را به صورت یک Window و خارج از محیط مرورگر اجرا کنیم.می توانیم آن را از منو استارت اجرا کنیم. می توانیم push notification ها را حتی زمانی که کاربر درحال کار با برنامه نیست دریافت کنیم. می توانیم به صورت پس زمینه برنامه را آپدیت کنیم و ... . به تصویر زیر نگاه کنید. می توانیم برنامه را به راحتی نصب کنیم:



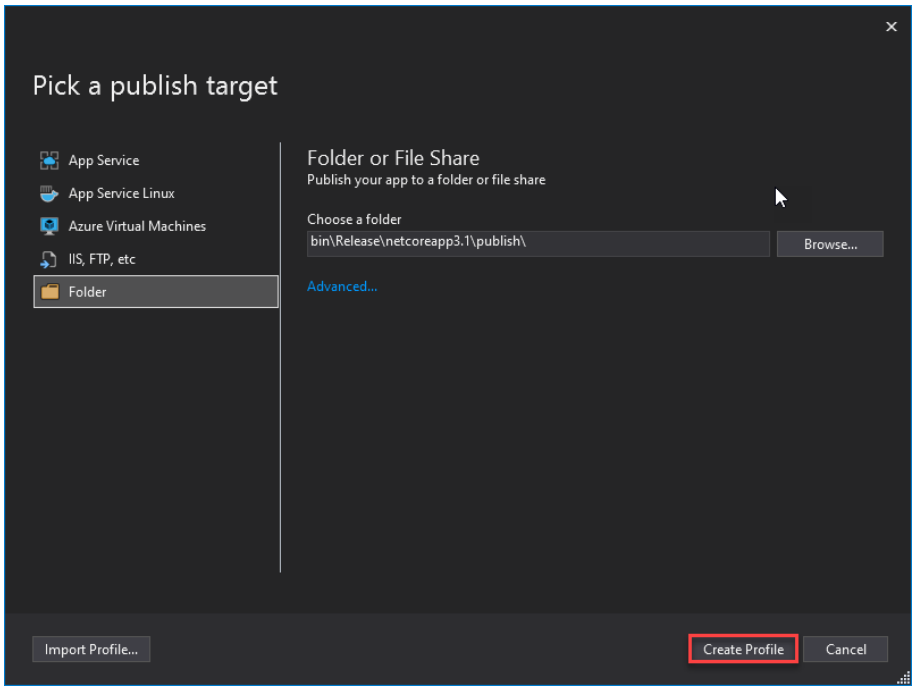
بعد از نصب، برنامه در Window جدید باز می شود. حتی می توانید آن را از منو start اجرا کنید. از طریق منو تنظیمات که در تصویر زیر مشخص شده، میتوانید آن را uninstall کنید.



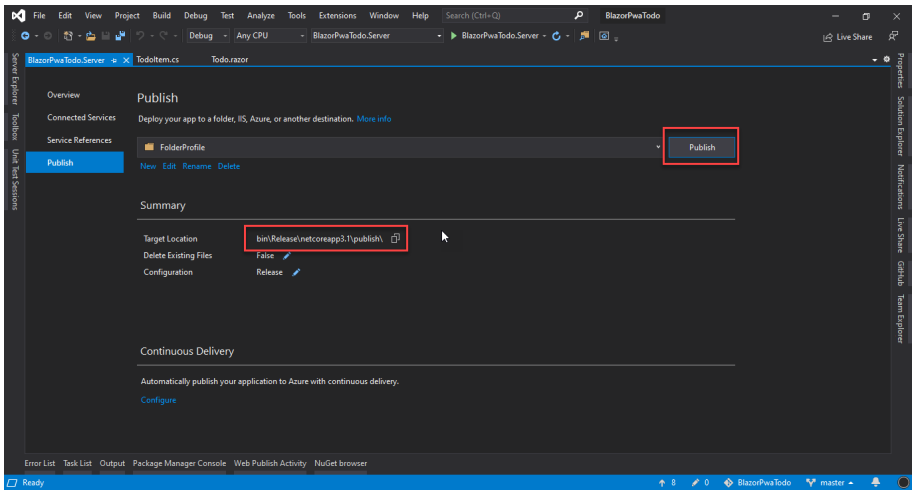
برخی از تنظیمات مثل title و آیکن صفحه از فایل manifest.json قابل ویرایش است.

۷.۴ بررسی نسخه آفلاین PWA

به صورت پیشفرض، امکان لود آفلاین در PWA فعال هست. کاربران برای بار اول باید برنامه را به صورت آنلاین دریافت کنند. مرورگر به صورت خودکار موارد لازم برای اجرای آفلاین و کش را ذخیره می کند.
هنگام توسعه پروژه PWA با Blazor این موارد لازم و کش ها ذخیره نمی شوند تا توسعه دهنده در هر بار آپدیت، تغییرات اعمال شده را ببینید. بنابراین نمی توانید درحالت توسعه نسخه آفلاین را ببینید و این امکان فقط در نسخه published قابل استفاده هست. پس برای تست نسخه آفلاین باید اپلیکیشن را publish کنیم.
برای این کار در Solution Explorer بر روی پروژه BlazorPwaTodo.Server کلیک راست کنید و روی publish کلیک کنید. همانند تصویر زیر گزینه Publish to Folder را انتخاب کرده و روی Create Profile کلیک کنید

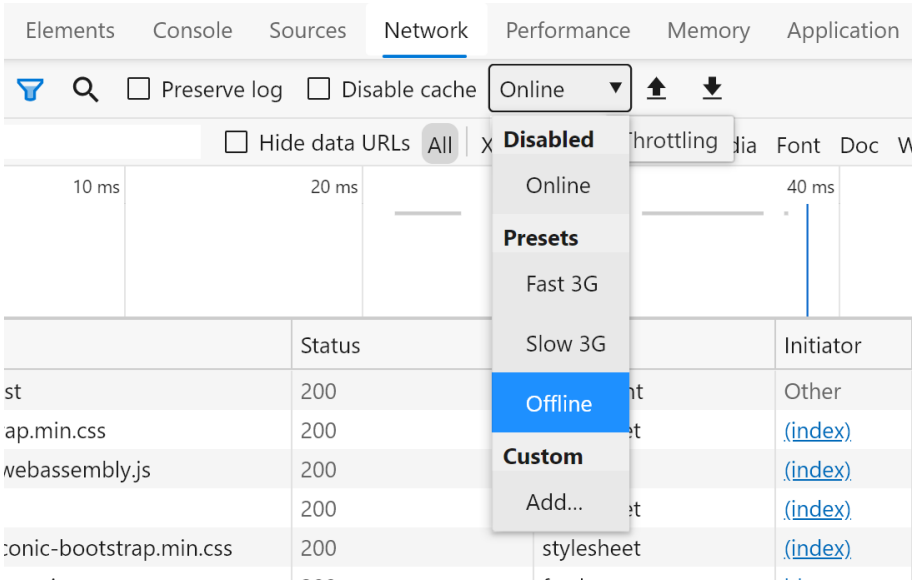


سپس در ادامه، روی Publish کلیک کنید. در صورتی که تنظیمات را تغییر نداده باشید فایل Publish شده در محل ذخیره سازی پروژه در مسیر bin/Release/netcoreapp۳.۱/publish قرار دارد:



حال، در محل Publish می توانید برنامه را با استفاده از فایل BlazorPwaTodo.Server.exe اجرا کنید یا دستور dotnet BlazorPwaTodo.Server.dll از طریق cmd یا powershell در همان محل

Publish اجرا کنید تا برنامه اجرا شود. اکنون برنامه در لوکال با پورت ۵۰۰۱ و ۵۰۰۰ در دسترس هست. کافی است آدرس https://localhost:۵۰۰۱ را در مرورگر وارد کنید(ترجیحا کروم). برای تست اجرای آفلاین طبق تصویر زیر، در مرورگر کروم حالت آفلاین را فعال کنید:



۸.۴ خلاصه و جمع بندی

در این بخش، یک اپ Todo ساده PWA با Blazor ایجاد کردیم. سپس اجرای آن را به صورت تحت ویندوز و همچنین آفلاین تست کردیم. PWA در Blazor امکانات زیادی را برای ما فرهم کرده است که در بخش های بعدی بعدی به آن می پردازیم.

موفق و مؤید باشید

منابع

http://www.tahlildadeh.com/

https://www.radcom.co/

http://provid.ir/

https://sourceiran.com/

https://soft۹۸.ir/

https://docs.microsoft.com/

https://softamoz.com/

https://www.programmingwithwolfgang.com/

https://www.zoomit.ir/

https://github.com/softamoz/BlazorPwaTodo